REPORT – COMPUTER VISION AND IMAGE PROCESSING - 19/02/2018

Taurone Francesco 834089 Automation Engineering

Project:

Deep learning application for face recognition

In collaboration with T3LAB

INTRODUCTION

This project aims the recognition via webcam stream of faces for a security purposed application. The starting point is an already trained network, part of FaceNet system [1], whose structure is going to be described later, as well as a GitHub project based on the same system [2] as a code reference. The solution presented is indeed intended as a starting point for further development.

Various packages were imported for the correct implementation of the program, among which OpenCV and Scikit. The code is written in python.

THE WORKING PRINCIPLE

The general cascade of events taking place in the application is the following. In the next sections, more details are going to be provided.

The image acquired by the webcam is analysed, looking for faces. Once the faces have been detected, the corresponding portions of the image are fed to the neural network, which returns the relative descriptors. The descriptors are then associated to a class, where each class is the ID number of a person among the ones present in the Dictionary.txt. The resulting names and frames of the faces are then drawn on the image and showed on video.

THE NETWORK

The network used in FaceNet is a CNN of 22 layers, having as output a descriptor of 128 bits trained with a triplet loss function aiming to minimize the distance of descriptors for similar images, while maximizing it for images with different subjects. In order to focus on recognition rather than on the network, further details on the network are just referenced [3].

In the GitHub template, the trained weights associated with each layer of the network were singularly loaded back to the network at the beginning of each execution. For the sake of speed, a model called “net_model.h5” has been obtained in substitution of these weights, so that the initialization of the net is faster.

SVM – SUPPORTED VECTOR MACHINE

The descriptors from the network summarize the important features of the detected faces in the image with a very small set of numbers. In order to recognize the persons in the image these descriptors have to be assigned a class based on their characteristics. At the very beginning, this task was done looking for the minimum Euclidean distance between database and image descriptors according to the FaceNet system, but the results were not satisfactory. Therefore, an SVM method from Scikit called SVC [4] has been deployed. This basically assigns portions of the descriptor space to the different classes based on a training set, where various images of each person are used to divide the 128-dimension space in sections, one for each class.

The training images are searched in ". /images_video/", where each frame was extracted from a recorded video using the code in "frameFromVideo.py". The current training set is composed of around 200 training images per class with less than 10 classes in total, which means less than 10 different identities to be recognized. The trained SVM object, for the sake of speed as well, is saved after training and loaded at the beginning of each next execution, since the training needs around 10 minutes with the available equipment. (No GPU, i3 processor)

THE RECENT_ID LIST

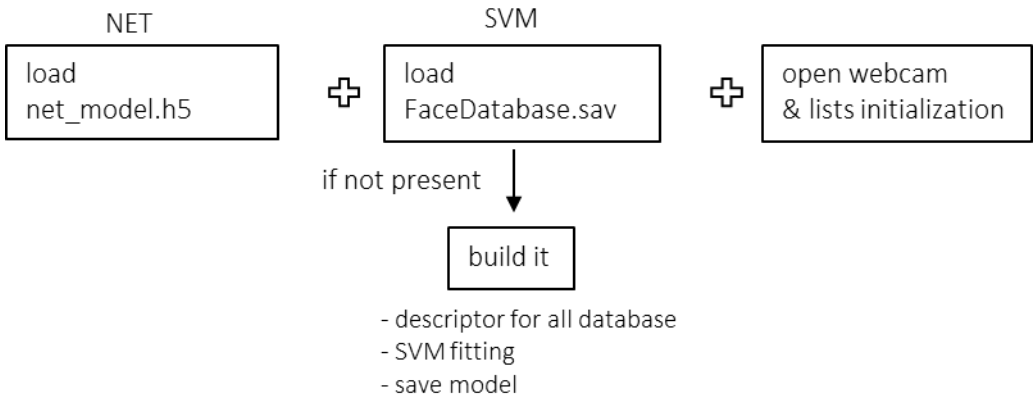
Although during the recognition process we strive for the achievement of a continuous correct identification, it is possible to encounter some isolated misidentifications due to a whole variety of reasons. In order to filter out these spurious results, the use of a list of recent identification might come in handy. The idea is based on the fact that for a video stream, the image currently analysed and the ones considered immediately before should be similar, therefore producing comparable results in terms of identification. Then by comparing the recent past with the present we might ideally be able to identify some erroneous results, substituting them with the verified output of the previous images.

This idea has yielded an increase in robustness but lacks precision in some known cases, that are going to be discussed during the analysis of the relative coding. Some proposals for an improved robustness are discussed as well.

FROM INPUT TO OUTPUT

Following the steps in the code and the nomenclature in the figures, the sequence of actions is analysed:

1- Initialization:

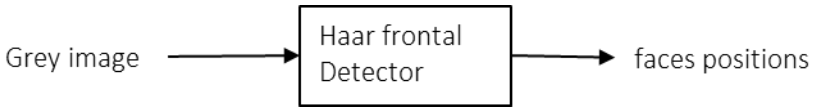


The model of the network is loaded at the very beginning, called “net_model.h5”, as well as the SVM model “FaceDatabase.sav”. If the latter is not present, there are two possible reasons: the first one is that it has never been created, the second that the training set of images and/or the entries of the dictionary have been modified, then the model was deleted so to be updated by building it again. In case the .sav file is not found, a new one is built automatically.

The building process for the SVM model is in 2 steps. First the function `prepare_database_for_svm(...)` is called, where each image in the folder is passed to the detector, the face (should be one) is identified and cropped, then passed through to the network whose job is to return the associated descriptor. Each image is named in the fashion “%d_%d”, where the first integer is the ID of the person in the picture, and the second is the image number. In this way each image is labelled as belonging to a particular person, so we end up having a long series of descriptors, each one associated with a class. The second step is the creation of the SVM model itself based on the obtained information.

Now the main function `webcam_face_recognizer(...)` can be called, which opens the stream from webcam and the relative windows, as well as initializing the list of the recent recognized identities with ‘-1’ (these will become helpful in the following stages).

2- Detection:

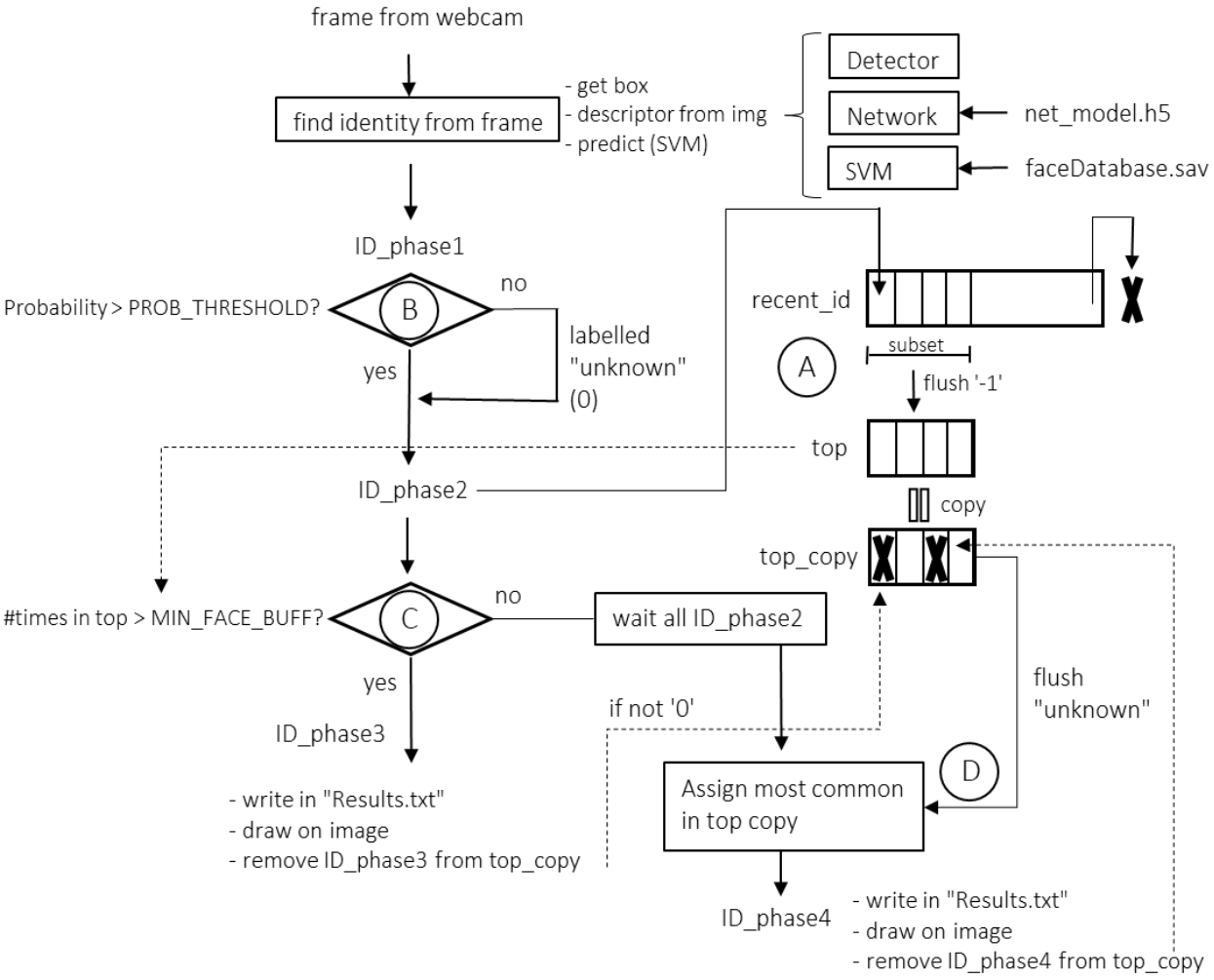


Detection was being studied by a different student in T3LAB during the development of this project. Therefore, as a temporary substitute, an OpenCV function called `haarcascade_frontalface_default` was used, which yields the drawback of being able to detect, then recognize, only frontal faces. The developed recognition system however is not limited to frontal images, so that once the two projects (detection + recognition) are merged, a more complete solution is obtained. The current set of images used for training the SVM is mainly of frontal images, so it would be necessary to update the database of labelled images as well.

3- Recognition

The Haar detector returns the position of the detected faces in the image from a greyed version of it. Each of these detected faces go under the recognition process. Before this however, a subset of the most recent identifications is selected from `Recent_ID` and called `top` (**A**), so to be used for processing the image as discussed previously. The more faces are detected in the current image, the longer is the selected subset, since the previous images are likely to have produced more useful identities.

This `top` list with the previous identities is flushed of the ‘-1’, whose meaning is “no identity”. These values are instrumental for filling the list at the beginning of execution, when no identity has been found yet, or after a period of length `FLUSH_BUFFER` with no faces in the image. A series of empty images might mean that the main actors are changing, so that the subjects detected in the future images are going to be different from the previous ones. For avoiding the past identities (that are now likely to be wrong) to influence the current recognitions, the buffer is filled with ‘-1’.



For each face in the images an initial guess on the identity is stored in the variable `identity` via `find_identity_svm(…)` . For clarity, at this stage we refer to these identities as ID_phase1. For doing this, the function calls `get_box(…)` for cropping the region of interest, computes the encoding through the neural network and guesses the identity using the SVM `model.predict`. Here is the first filter for the guesses, which is a threshold based on the probability of the instance to belong to that particular class, computed via `model.predict_proba(..)`. The filtered-out guesses are flagged as unknown with the ID '0'. These new identities are referred as ID_phase2. (**B**)

The ID_phase2 are now added to the `recent_ID` list, and the last entry of this list is eliminated, so to maintain the same length. It has to be noted that ID_phase2 are not used as recent identifications for the current image, since the comparison has to be based only on the previous images. This is why the list `top` is selected before any processing on the image (**A** before **B**)

Now the guesses are filtered again, this time looking at the recent entries of `top`, meaning at the most recent identities of the previous images. It's likely that if a person is present in the current image, it was present also in the previous ones. Therefore, so to filter out wrong temporary identifications, the ID_phase2 that are present in the top list at least MIN_FACE_BUFF times are considered to be authentic, whereas the ones that do not respect this requirement have to wait for all ID_phase2 to be processed (**C**).

The confirmed identities, referred now as ID_phase3, are written in the "Results.txt" file, the corresponding name in the dictionary is drawn in the picture, and all the entries matching with that validated ID are removed from a copy of `top`, used so to keep the original intact, called `top_copy`.

All the zeros in top_copy are deleted since we don't want those to affect the assignment of the invalid ID_phase2.

Once all the zeros representing unknowns in the list are deleted from top_copy and after waiting for all the ID_phase2 to be processed, the invalid ID_phase2 are assigned a new identity called ID_phase4, in particular the most common taken from `top_copy`. This is why ID_phase3 are removed from it, so to ensure that no ID_phase4 is equal to an ID_phase3. In this way we ensure that one ID appears only once per image on screen (**D**). The exception to this is for ID_phase3 being a '0', namely "Unknown", that are not deleted once found since multiple unknown faces might be found in a single image at the same time.

The reason for assigning the most common entries in `top_copy` is that these identities might correspond to figures not present in the image anymore, or to faces currently in the image but misclassified. In the latter case, assuming a small number of wrong assignments, this strategy filters some of these classification errors. ID_phase4 are written as results, removed from `top_copy` and drawn in the image as well.

It has to be pointed out that '-1' in (**A**) and '0' in (**D**) are flushed from the list so that they cannot become ID_phase4 at this stage.

Furthermore, identities flagged as unknown are stored in `recent_id` like any other ID so to ensure that if a person that is not in the database keeps appearing in the image, it fulfils the condition on MIN_FACE_BUFF (**C**) and is labelled "unknown", rather than being invalid and assigned with one in `top_copy`.

This method has shown to improve the robustness especially for wrong isolated classifications. It fails however in assigning back these common IDs when multiple faces are in the wrong class. If, for example, Anna and Luca were assigned the classes "Paolo" and "Giovanni" respectively, this algorithm might recognize the wrong classification in (**C**) but then assign the label "Anna" to Luca and "Luca" to Anna, given that no control is made on the location of the previous faces. This further development might significantly increase the overall reliability of the process.

The very same coordinate-based control might improve the performances of this method also for the cases in which the ID_phase2 marked as invalid is already present in the image, such as Anna being assigned with "Paolo" while Paolo is in the image. In this case, depending on the image, the label "Paolo" might be kept by Anna instead of belonging to Paolo.

After these steps the frame rate is calculated as the inverse of the time needed for processing the image and then drawn on it at the upper left corner.

The last check is on the variable `empty_frame`, that counts the number of consequent frames in which no face was detected. As soon as a certain FLUSH_BUFFER number is reached, `recent_id` is flushed and filled with '-1'. This is again for making sure that the last people present in the image do not influence the recognition of the current and future ones if a relatively long period of time passed with no people in the scene.

The image is then shown alongside the original one.

PREVIOUS STRATEGIES

Rather than relying on a SVM techniques, FaceNet achieves recognition by comparing descriptors using the Euclidean distance. More specifically, the descriptor of the frame from webcam is compared with the descriptors in the database, and the one that minimizes the distance is labelled as a match. The database in this case is a list of descriptors: each one of those is obtained from a single image of reference, one for each person. Even in this case, if the minimum distance found after searching the database is above a certain threshold, the face is marked as Unknown.

This method was the one initially deployed during the development so to follow the standard FaceNet implementation. However, after few days of tests and adjustments, the results were still not satisfactory. This is the main reason why the SVM approach has taken place over the Euclidean distance. Some of the previous functions based on this approach are still in the code just as reference.

RESULTS

The results were stored for each execution in "Results.txt", in the format of a series of IDs where each row represents a frame of the stream, and the relative identities are separated by a tab.

Here an example of recognition with "Francesco" as the single subject in the image, while moving, for longer and shorter executions:

Dictionary		times	%	times	%	times	%	times	%	times	%	RESULTS
Unknown	0	23	13%	7	5%	3	4%	205	15%	20	9%	9%
Andrea	1	0	0%	0	0%	0	0%	3	0%	0	0%	0%
Federica	3	7	4%	5	4%	7	9%	11	1%	4	2%	4%
Francesco	4	152	84%	119	88%	69	87%	1084	78%	201	88%	85%
Buratti	6	0	0%	5	4%	0	0%	87	6%	3	1%	2%
TOT		182		136		79		1390		228		

"Francesco" was correctly recognized around 85% of the times.

The percentage of unknowns might be adjusted by choosing different thresholds for labelling. The one used for these experiments was based on empirical evaluations.

Here with "Francesco" and "Elisa", both in the scene. Given that both subjects were in the image at the same time for the whole experiment, a perfect combination of detection and recognition would yield as result 50% for both "Francesco" and "Elisa":

Dictionary		free		free		free		static		moving		RESULTS
Unknown	0	11	9%	7	6%	70	28%	5	2%	74	49%	19%
Andrea	1	0	0%	0	0%	0	0%	0	0%	4	3%	1%
Federica	3	0	0%	0	0%	0	0%	0	0%	0	0%	0%
Francesco	4	43	35%	38	33%	108	43%	114	48%	34	22%	36%
Buratti	6	0	0%	0	0%	0	0%	0	0%	0	0%	0%
Elisa	8	68	56%	69	61%	74	29%	119	50%	40	26%	44%
TOT		122		114		252		238		152		

Analysing free, static and moving scenarios, an overall misclassification rate of 20% was found, which complies with the previous experiment.

As further test, a video is attached showing different trials. In particular, some have a name at the bottom left corner of the blue frame, which represents ID_phase2, so before the strategy with `recent_id` is applied (C). The top left names instead are ID_phase3 and ID_phase4, so to prove the effectiveness of the strategy. A person in the video was left out of the database so to check the behaviour for unknowns. A folder with images for identity reference are attached as well. Moreover, some video sections were increased in speed, then in these cases the frame rate printed on video should not be considered.

It has been noted that frame rate heavily depends on faces being present in the image or not, probably because of the passage through the net.

CONCLUSIONS & FUTURE DEVELOPMENT

This system has proved to be somewhat effective, however needs further refinements on many aspects for a security application. Among these:

- Use a different Neural Network. The used one was proposed in combination with the Euclidean norm technique, providing a small set of numbers as descriptors. However, for this application, different descriptors and SVM oriented network might improve the overall performances;
- Implementation of the detector from the student in T3LAB for a more complete solution;
- The parameters in the code like the length of lists and the threshold levels might be optimized for better performances;
- Add a coordinate-based stage (C) so to check the consistency of new identifications based on both their location and the past images. The idea might be to assign ID_phase4 according to the faces identified in previous images but only looking at a neighbourhood of the current face position, so to avoid inverted assignments in (C);
- In order to improve the intensity invariance property of the system, and equalization of the channels was initially implemented. However, it seemed to worsen the results in terms of recognition, so the code was just left commented as reference. Nonetheless, scale and intensity invariances for recognition are strongly suggested as further developments.

BIBLIOGRAPHY

[1] Medium, «Free Code Campus,» [Online]. Available: https://medium.freecodecamp.org/making-your-own-face-recognition-system-29a8e728107c.

[2] Skuldur, «GitHub,» [Online]. Available: https://github.com/Skuldur/facenet-face-recognition.

[3] Schroff, Kalenichenki e Philbin, «Google,» [Online]. Available: https://arxiv.org/pdf/1503.03832.pdf.

[4] SciKit, «SVM - SVC,» [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.